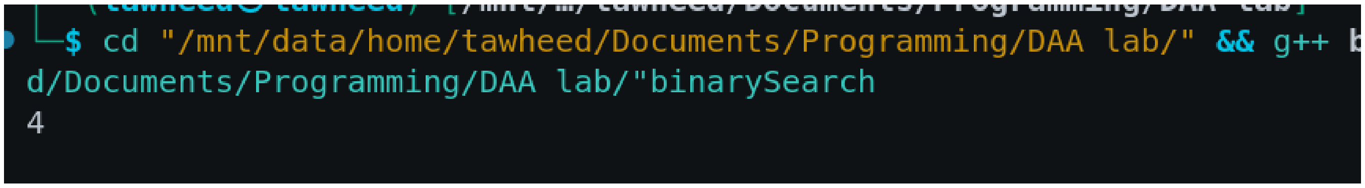
Tavaheed tariq

Enrollment Number: 2022BITE008

Design and Analysis of Algorithms Lab

Binary Search

```cpp
#include <iostream>
#include <bits/stdc++.h>
using namespace std;

int binary_search(int arr[], int start, int end, int key){
    int mid = start + (end- start)/2;
    if(start > end)
        return -1;
    if(arr[mid] == key)
        return mid;
    else if(arr[mid] > key)
        return binary_search(arr, 0, mid-1, key);
    else
        return binary_search(arr, mid+1, end, key);
}

int main(){
    int n = 6,key = 8;
    int arr[] = {1,2,4,6,8,9};
    cout << binary_search(arr,0, n, key);
}
```



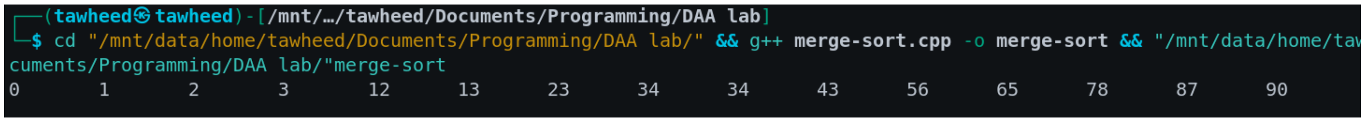
```
└─$ cd "/mnt/data/home/tawheed/Documents/Programming/DAA lab/" && g++ b
d/Documents/Programming/DAA lab/"binarySearch
4
```

Merge Sort

```cpp
// merge sort using divide and conquer

#include<iostream>
```

```
#include<vector>
using namespace std;

void merge(vector<int> &arr, int s, int e){
    //divide arr in two parts
    int mid = s + (e-s)/2;
    int len1 = mid - s +1, len2 = e-mid;
    int *first_arr = new int[len1];
    int *second_arr = new int[len2];


    //add elements in arrays
    int mainArray_index = s;
    for(int i = 0; i < len1; i++){
        first_arr[i] = arr[mainArray_index++];
    }
    for(int i =0; i < len2; i++){
        second_arr[i] = arr[mainArray_index++];
    }

    //merge two sorted arrays
    int index1 = 0, index2 = 0;
    mainArray_index = s;
    while(index1 < len1 && index2 < len2){
        if(first_arr[index1] < second_arr[index2])
            arr[mainArray_index++] = first_arr[index1++];
        else
            arr[mainArray_index++] = second_arr[index2++];
    }
    while(index1 < len1){
        arr[mainArray_index++] = first_arr[index1++];
    }
    while(index2 < len2){
        arr[mainArray_index++] = second_arr[index2++];
    }
    delete []first_arr;
    delete []second_arr;
}

void mergeSort(vector<int> &arr, int s, int e){
    if(s >= e)
        return;
    int mid = s + (e-s)/2;
    mergeSort(arr,s,mid);
    mergeSort(arr,mid+1, e);
```

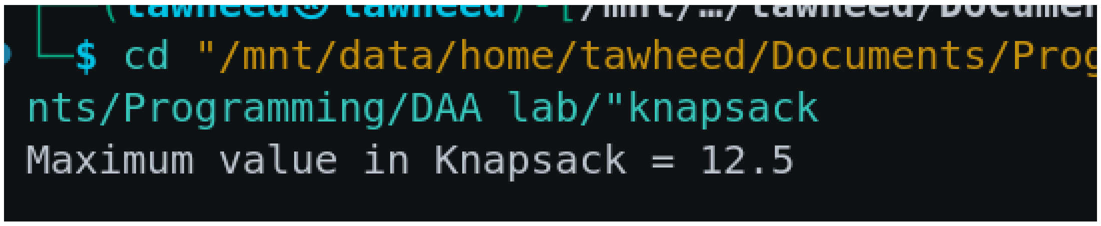
```cpp
    merge(arr,s,e);
}
```



```
┌──(tawheed㉿tawheed)-[/mnt/…/tawheed/Documents/Programming/DAA lab]
└─$ cd "/mnt/data/home/tawheed/Documents/Programming/DAA lab/" && g++ merge-sort.cpp -o merge-sort && "/mnt/data/home/taw
cuments/Programming/DAA lab/"merge-sort
0       1       2       3       12      13      23      34      34      43      56      65      78      87      90
```

Fractional Knapsack

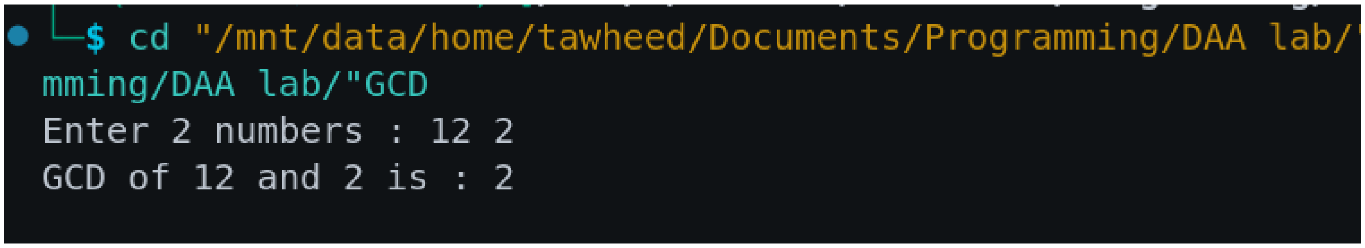
```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Item {
    int weight;
    int value;
};

// Comparison function to sort items by value-to-weight ratio
bool compare(Item a, Item b) {
    double r1 = (double)a.value / a.weight;
    double r2 = (double)b.value / b.weight;
    return r1 > r2;
}

double fractionalKnapsack(int W, vector<Item>& items) {
    sort(items.begin(), items.end(), compare);

    double totalValue = 0.0;
    for (const auto& item : items) {
        if (W == 0) break;
        if (item.weight <= W) {
            W -= item.weight;
            totalValue += item.value;
        } else {
            totalValue += item.value * ((double)W / item.weight);
            W = 0;
        }
    }
    return totalValue;
}

int main() {
    int W = 50;
    Item a = {10 , 40};
    vector<Item> items = {{60, 10}, {100, 20}, {120, 30}};

    double maxValue = fractionalKnapsack(W, items);
    cout << "Maximum value in Knapsack = " << maxValue << endl;

    return 0;
}
```



```
  (tawheed㉿tawheed)-[/mnt/…/tawheed/Documen
└─$ cd "/mnt/data/home/tawheed/Documents/Prog
nts/Programming/DAA lab/"knapsack
Maximum value in Knapsack = 12.5
```

GCD

```cpp
#include <iostream>
using namespace std;

int gcd(int a, int b){
    if(b == 0){
        return a;
    }
    return gcd(b, a%b);
}

int main(){
    int a, b;
    cout << "Enter 2 numbers : ";
    cin >> a >> b;
    cout << "GCD of " << a << " and " << b << " is : " << gcd(a, b);
    return 0;
}
```

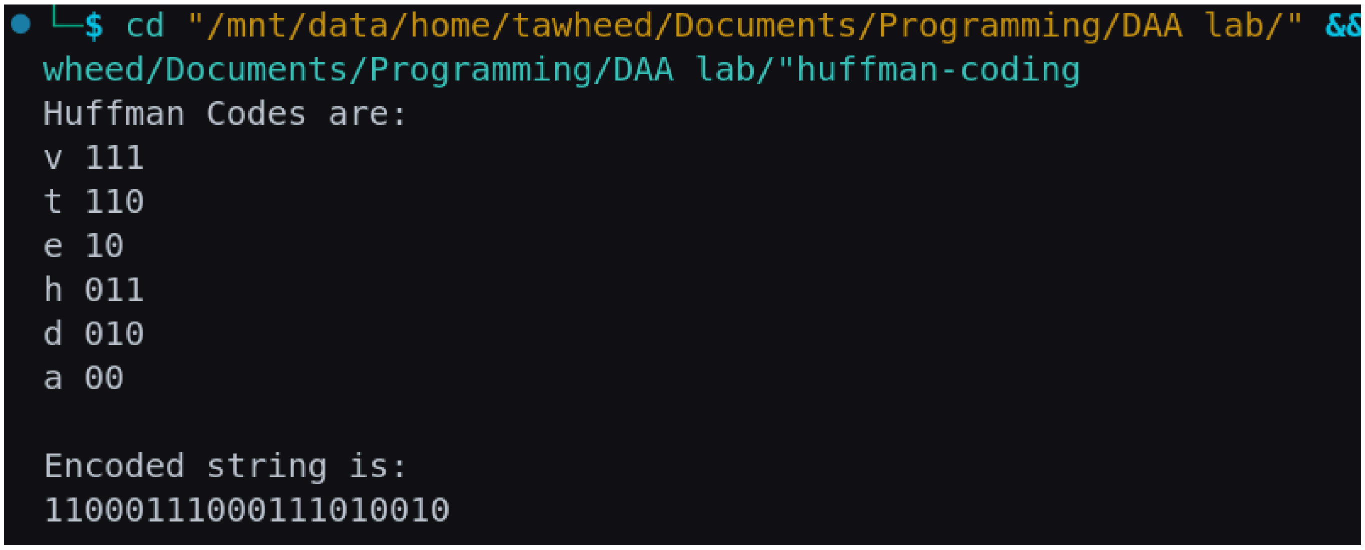


```
└─$ cd "/mnt/data/home/tawheed/Documents/Programming/DAA lab/"
mming/DAA lab/"GCD
Enter 2 numbers : 12 2
GCD of 12 and 2 is : 2
```

Huffman Coding

```cpp
#include <iostream>
#include <queue>
#include <unordered_map>
#include <vector>

using namespace std;

struct Node {
    char ch;
    int freq;
    Node *left, *right;

    Node(char ch, int freq) {
        left = right = nullptr;
        this->ch = ch;
        this->freq = freq;
    }

    Node(char ch, int freq, Node* left, Node* right) {
        this->ch = ch;
        this->freq = freq;
        this->left = left;
        this->right = right;
    }
};

struct compare {
    bool operator()(Node* l, Node* r) {
        return l->freq > r->freq;
    }
};

//traversing the tree and sorting the huffman code for each character
void encode(Node* root, string str, unordered_map<char, string>
&huffmanCode) {

    if (root == nullptr)
        return;

    if (!root->left && !root->right) {
        huffmanCode[root->ch] = str;
    }

    encode(root->left, str + "0", huffmanCode);
    encode(root->right, str + "1", huffmanCode);
}
```

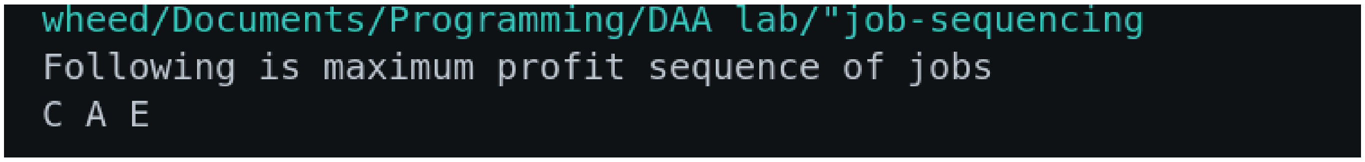
```
void buildHuffmanTree(string text) {
    unordered_map<char, int> freq;
    //find frequency of each character
    for (char ch : text) {
        freq[ch]++;
    }

    priority_queue<Node*, vector<Node*>, compare> pq;
    //add leaf node to priority queue so that we can extract two minimum
frequency nodes each time
    for (auto pair : freq) {
        pq.push(new Node(pair.first, pair.second));
    }
    //create a tree by extracting two minum nodes from priiotiy queue and if
only 1 node is present then it is the root node
    while (pq.size() != 1) {
        Node *left = pq.top(); pq.pop();
        Node *right = pq.top(); pq.pop();

        int sum = left->freq + right->freq;
        pq.push(new Node('\0', sum, left, right));
    }

    Node* root = pq.top();

    unordered_map<char, string> huffmanCode;
    encode(root, "", huffmanCode);

    cout << "Huffman Codes are:\n";
    for (auto pair : huffmanCode) {
        cout << pair.first << " " << pair.second << '\n';
    }

    string str = "";
    for (char ch : text) {
        str += huffmanCode[ch];
    }

    cout << "\nEncoded string is:\n" << str << '\n';

}

int main() {
    string text = "tavaheed";
    buildHuffmanTree(text);
    return 0;
}
```



```
└─$ cd "/mnt/data/home/tawheed/Documents/Programming/DAA lab/" &&
wheed/Documents/Programming/DAA lab/"huffman-coding
Huffman Codes are:
v 111
t 110
e 10
h 011
d 010
a 00

Encoded string is:
11000111000111010010
```

Job Sequencing

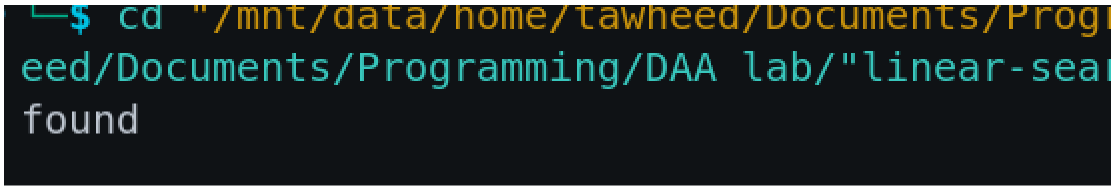
```cpp
#include <iostream>
#include<bits/stdc++.h>
using namespace std;

struct Job
{
    char id;
    int dead;
    int profit;
};

bool comparison(Job a, Job b)
{
    return (a.profit > b.profit);
}

void printJobScheduling(Job arr[], int n)
{
    sort(arr, arr + n, comparison);

    int len = -1;
    for(int i =0; i < n; i++){
        if(arr[i].dead > len){
            len = arr[i].dead;
        }
    }
    unordered_map<int, bool> mp;
    int result[len];
    for(int i = len-1; i >= 0; i--){
        result[i] = -1;
        for(int j = 0; j < n; j++){
            if(arr[j].dead > i){
                if(mp.find(j) == mp.end()){
                    result[i] = j;
                    mp[j] = true;
                    break;
                }
            }
        }

    }

    for (int i = 0; i < len; i++)
    {
        cout << arr[result[i]].id << " ";
    }
}

int main()
{
    Job arr[] = {{'A', 2, 100}, {'B', 1, 19}, {'C', 2, 27}, {'D', 1, 25},
{'E', 3, 15}};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Following is maximum profit sequence of jobs\n";
    printJobScheduling(arr, n);
    return 0;
}
```



```
wheed/Documents/Programming/DAA lab/"job-sequencing
Following is maximum profit sequence of jobs
C A E
```

Linear Search

```
#include<iostream>
using namespace std;
bool search(int *arr, int n, int key);
int main(){
    int n = 7, arr[n] = {1,2,3,4,5,6}, key = 5, result;
    int res = search(arr, n , key);
    if(res == 1)
            cout << "found" << endl;
        cout << "not found"<< endl;
        return 0;
}

bool search(int *arr, int n, int key){
    for(int i =0 ; i < n; i++){
        if(*(arr+i) == key)
            return 1;
    }
    return 0;
}
```

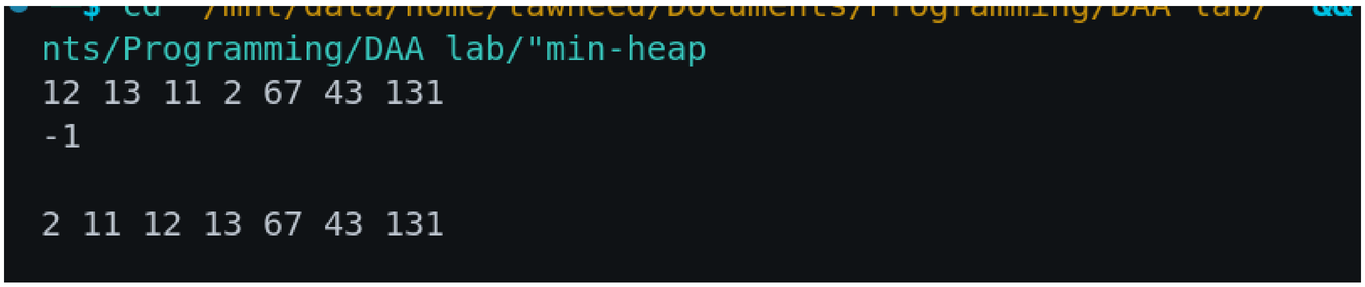


```
└─$ cd "/mnt/data/home/tawheed/Documents/Prog
eed/Documents/Programming/DAA lab/"linear-sea
found
```

Minimum Heap

```
#include<iostream>
#include<bits/stdc++.h>
using namespace std;

// 1 based indexing
class heap{
    public:
    int size;
    int *arr;
    heap(){
        size = 0;
        arr = new int[100000];
    }
    void insetion(int info){
        size++;
        int index = size;
        arr[size] = info;
        while(index > 1){
            int parent = index/2;
            if(arr[parent] > arr[index]){
                swap(arr[parent], arr[index]);
                index = parent;
            }else{
```

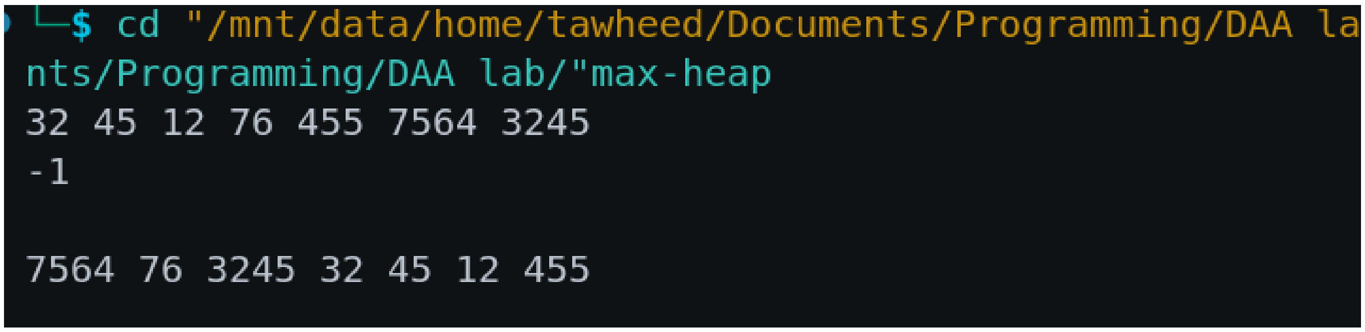
```cpp
                return;
            }
        }
    }

//nodes after n/2 of array are leaf nodes so traverse from
//n/2 to 0 and heapify every element
    void heapify(int i ){
        int smallest = i, left = 2*i , right = 2*i +1;

        if(left <= size && arr[left] < arr[smallest]){
            smallest = left;
        }
        else if(right <= size && arr[right] < arr[smallest]){
            smallest = right;
        }
        if(smallest != i){
            swap(arr[i], arr[smallest]);
            heapify(smallest);
        }
    }
};



int main(){
    heap h;
    int x;
    while(1){
        cin >> x;
        if(x == -1)
            break;
        h.insetion(x);
    }
    cout << endl;
    for(int i = 1; i <= h.size; i++){
        cout << h.arr[i] << " ";
    }
    return 0;
}
```



```
~ $ cd "/mnt/data/home/tawheed/Documents/Programming/DAA lab/" &&
nts/Programming/DAA lab/"min-heap
12 13 11 2 67 43 131
-1

2 11 12 13 67 43 131
```

Maximum Heap

```
#include<iostream>
#include<bits/stdc++.h>
using namespace std;

// 1 based indexing
class heap{
    public:
    int size;
    int *arr;
    heap(){
        size = 0;
        arr = new int[100000];
    }
    void insetion(int info){
        size++;
        int index = size;
        arr[size] = info;
        while(index > 1){
            int parent = index/2;
            if(arr[parent] < arr[index]){
                swap(arr[parent], arr[index]);
                index = parent;
            }else{
                return;
            }
        }
    }

//nodes after n/2 of array are leaf nodes so traverse from
//n/2 to 0 and heapify every element
    void heapify(int i ){
        int largest = i, left = 2*i , right = 2*i +1;

        if(left <= size && arr[left] > arr[largest]){
            largest = left;
        }
        if(right <= size && arr[right] > arr[largest]){
            largest = right;
        }
        if(largest != i){
            swap(arr[i], arr[largest]);
            heapify(largest);
        }
    }
};


int main(){
    heap h;
    int x;
    while(1){
        cin >> x;
        if(x == -1)
            break;
        h.insetion(x);
    }
    cout << endl;
    for(int i = 1; i <= h.size; i++){
        cout << h.arr[i] << " ";
    }
    return 0;
}
```

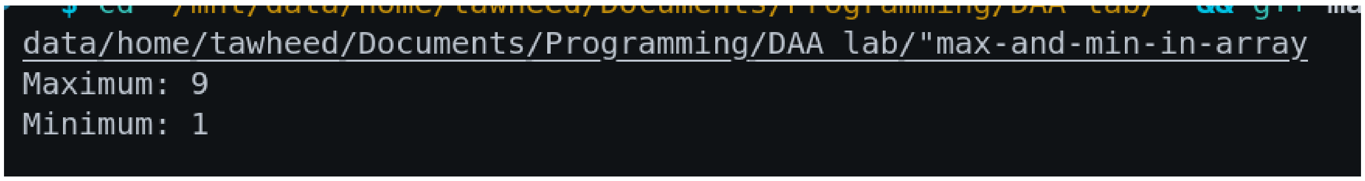


```
└─$ cd "/mnt/data/home/tawheed/Documents/Programming/DAA la
nts/Programming/DAA lab/"max-heap
32 45 12 76 455 7564 3245
-1

7564 76 3245 32 45 12 455
```

Max and Min in array

```
#include <iostream>
#include <vector>
#include <limits.h>

using namespace std;

pair<int, int> findMaxMin(const vector<int>& arr) {
```

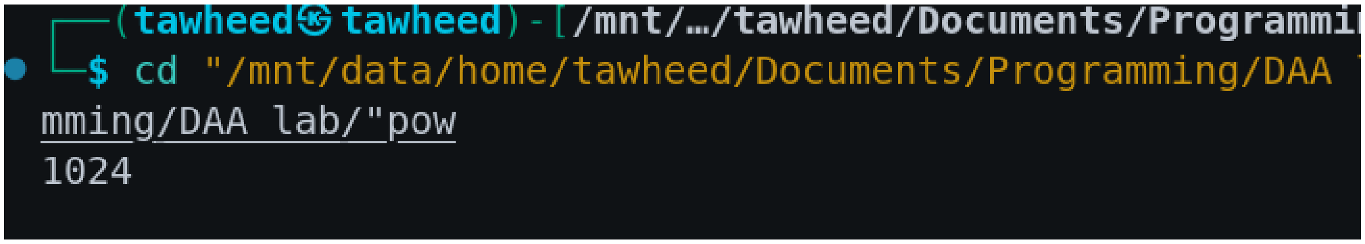
```cpp
    if (arr.empty()) {
        throw invalid_argument("Array is empty");
    }

    int maxVal = INT_MIN;
    int minVal = INT_MAX;

    for (int num : arr) {
        if (num > maxVal) {
            maxVal = num;
        }
        if (num < minVal) {
            minVal = num;
        }
    }

    return {maxVal, minVal};
}

int main() {
    vector<int> arr = {3, 5, 1, 8, 2, 9, 4};
    try {
        pair<int, int> result = findMaxMin(arr);
        cout << "Maximum: " << result.first << endl;
        cout << "Minimum: " << result.second << endl;
    } catch (const invalid_argument& e) {
        cerr << e.what() << endl;
    }

    return 0;
}
```



```
 $ cd "/mnt/data/home/tawheed/Documents/Programming/DAA lab/" && g++ ma
data/home/tawheed/Documents/Programming/DAA lab/"max-and-min-in-array
Maximum: 9
Minimum: 1
```

Power of Element

```cpp
#include<iostream>
#include<bits/stdc++.h>

using namespace std;

int power(int a, int n){
    if(n == 0)
        return 1;
    if(n == 1)
        return a;
    int x = power(a, n/2);
    if(n%2 == 0)
        return x*x;
    return x*x*a;
}

int main(){
    int a = 2, n = 10;
    cout << power(a, n);
}
```

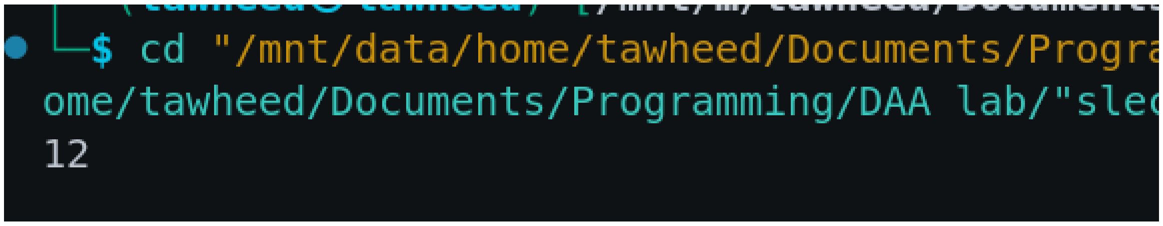


```
┌──(tawheed㉿tawheed)-[/mnt/…/tawheed/Documents/Programmi
└─$ cd "/mnt/data/home/tawheed/Documents/Programming/DAA 
mming/DAA lab/"pow
1024
```

Selection Procedure

```
// use selection procedure having an input --> unsorted array of integers
and k as an integer , find the kth smallest element

#include <iostream>
#include <vector>
using namespace std;

int partition(vector<int> &arr, int s, int e){
    int pivot = arr[e];
    int i = s-1;
    for(int j = s; j < e; j++){
        if(arr[j] < pivot){
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i+1], arr[e]);
    return i+1;
}

int kthSmallest(vector<int> &arr, int s, int e, int k){
```

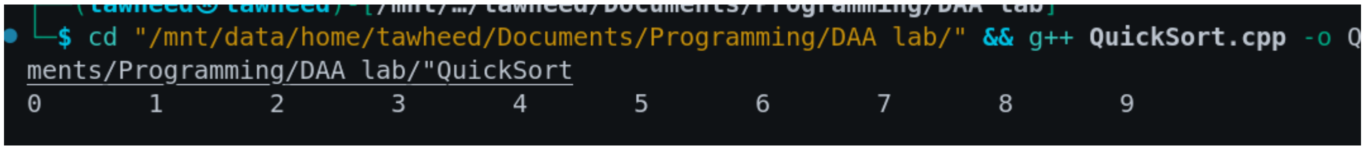
```cpp
    if(s <= e){
        int p = partition(arr, s, e);
        if(p == k-1)
            return arr[p];
        else if(p > k-1)
            return kthSmallest(arr, s, p-1, k);
        else
            return kthSmallest(arr, p+1, e, k);
    }
    return -1;
}

int main(){
    int n = 10;
    vector<int> arr = {12,2,34,23,56, 78, 3,0,1,43};
    int k = 4;
    cout << kthSmallest(arr, 0, n-1, k);
}
```



```
└─$ cd "/mnt/data/home/tawheed/Documents/Progra
ome/tawheed/Documents/Programming/DAA lab/"slec
12
```

Quick Sort

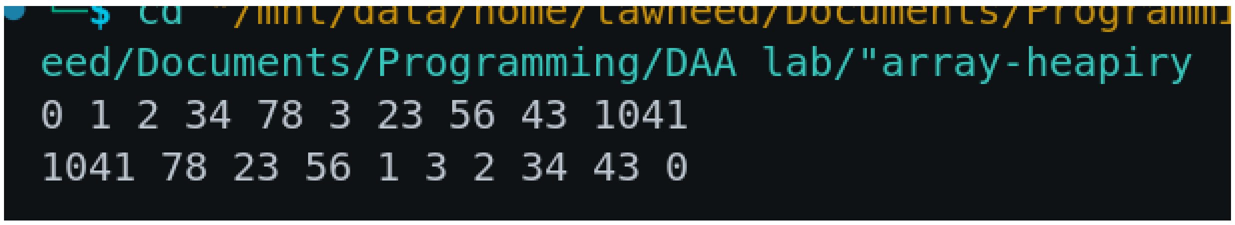
```cpp
#include<iostream>
#include <bits/stdc++.h>
using namespace std;
int partition(int arr[], int s, int e){
    int pivot = arr[s], count= 0;
    for(int i = s+1; i <= e; i++){
        if(arr[i] <= pivot)
            count++;
    }
    //taking pivot to its right position
    int pivotIndex = count +s;
    swap(arr[pivotIndex], arr[s]);

    //making left side of pivot <= pivot and right side of pivot >= pivot
    int i = s, j = e;
    while(i < pivotIndex && j > pivotIndex){
        while(arr[i] < pivot){
            i++;
        }
        while(arr[j] > pivot){
            j--;
        }
        if(i<pivotIndex && j > pivotIndex){
            swap(arr[i++], arr[j--]);
        }
    }
    return pivotIndex;
}

void quickSort(int arr[], int s, int e){
    if(s >= e)
        return;
    int p = partition(arr, s,e);
    quickSort(arr,s,p-1);
    quickSort(arr,p+1,e);
}

int main(){
    int arr[10] = {5,3,6,1,7,8,2,4,9,0}, n= 10;
    quickSort(arr,0,n-1);
    for(int i =0; i < n; i++){
        cout << arr[i] << "\t";
    }
}
```



```
  (tawheed㉿tawheed)-[/mnt/…/tawheed/Documents/Programming/DAA lab]
└─$ cd "/mnt/data/home/tawheed/Documents/Programming/DAA lab/" && g++ QuickSort.cpp -o Q
ments/Programming/DAA lab/"QuickSort
0       1       2       3       4       5       6       7       8       9
```

Array Heapify

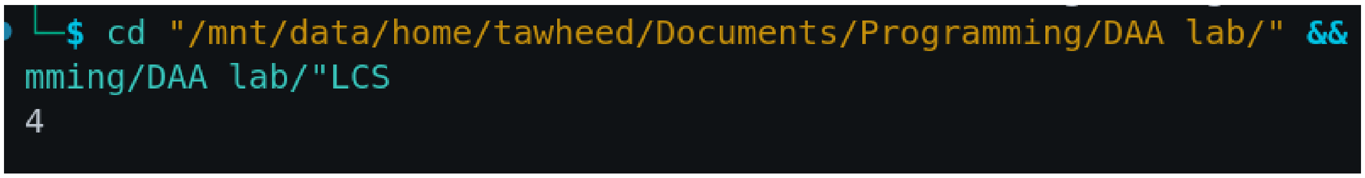
```
// create an array of n elements and then use min heapity and max heapity to
create a min heap and max heap respectively.

#include <iostream>
#include <vector>
using namespace std;

void min_heapify(vector<int> &arr, int i, int n){
    int smallest = i, left = 2*i, right = 2*i + 1;
    if(left <= n && arr[left] < arr[smallest]){
        smallest = left;
    }
    if(right <= n && arr[right] < arr[smallest]){
        smallest = right;
    }
    if(smallest != i){
        swap(arr[i], arr[smallest]);
        min_heapify(arr, smallest, n);
    }
}

void max_heapify(vector<int> &arr, int i, int n){
    int largest = i, left = 2*i, right = 2*i + 1;
    if(left <= n && arr[left] > arr[largest]){
        largest = left;
    }
    if(right <= n && arr[right] > arr[largest]){
        largest = right;
    }
    if(largest != i){
        swap(arr[i], arr[largest]);
        max_heapify(arr, largest, n);
    }
}


int main(){
    int n = 10;
    vector<int> arr = {12,2,34,23,56, 78, 3,0,1,43};
    for(int i = n/2; i >= 1; i--){
        min_heapify(arr, i, n);
    }
    for(int i = 1; i <= n; i++){
        cout << arr[i] << " ";
    }
    cout << endl;
    for(int i = n/2; i >= 1; i--){
        max_heapify(arr, i, n);
    }
    for(int i = 1; i <= n; i++){
        cout << arr[i] << " ";
    }
}
```



```
└─$ cd "/mnt/data/home/tawneed/Documents/Programmi
eed/Documents/Programming/DAA lab/"array-heapiry
0 1 2 34 78 3 23 56 43 1041
1041 78 23 56 1 3 2 34 43 0
```

Least Common sequence

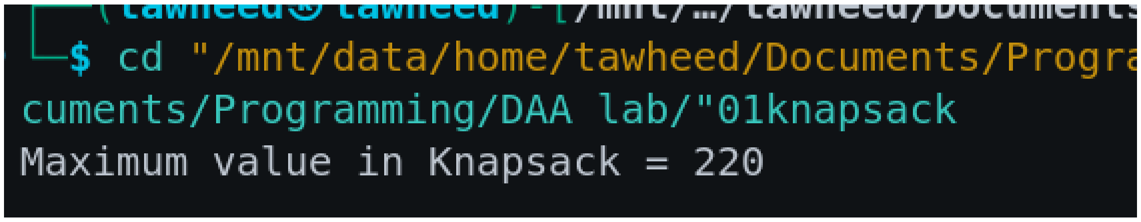
```cpp
#include<iostream>
#include<bits/stdc++.h>

using namespace std;

int LCSLength(string s1, string s2,int m, int n, vector<vector<int>> &dp){
    if(m == 0 || n == 0){
        return 0;
    }
    if(dp[m][n] != -1){
        return dp[m][n];
    }
    if(s1[m-1] == s2[n-1]){
        dp[m][n] = 1 + LCSLength(s1, s2, m-1, n-1, dp);
    }
    else{
        dp[m][n] = max(LCSLength(s1, s2, m-1, n, dp), LCSLength(s1, s2, m,
n-1, dp));
    }
    return dp[m][n];
}

int main(){
    string s1 = "AGGTAB";
    string s2 = "GXTXAYB";
    //here the longest common subsequence is "GTAB" of length 4
    int m = s1.length();
    int n = s2.length();
    vector<vector<int>> dp(m+1, vector<int>(n+1, -1));

    int len = LCSLength(s1, s2, m, n, dp);
    cout<<len<<endl;
    return 0;
}
```



```
└─$ cd "/mnt/data/home/tawheed/Documents/Programming/DAA lab/" &&
mming/DAA lab/"LCS
4
```

0/1 Knapsack

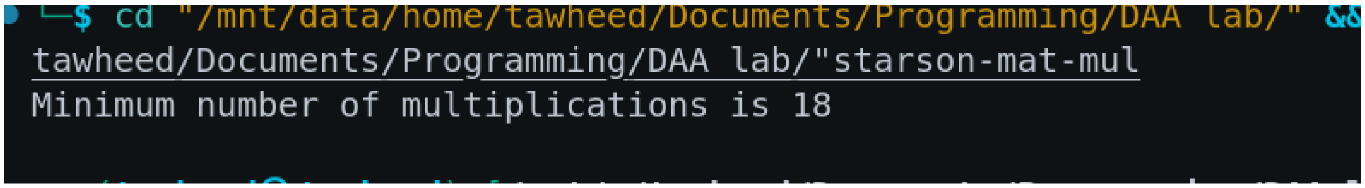
```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int knapsack(int W, const vector<int>& weights, const vector<int>& values,
int n) {
    vector<vector<int>> dp(n + 1, vector<int>(W + 1, 0));

    for (int i = 1; i <= n; ++i) {
        for (int w = 0; w <= W; ++w) {
            if (weights[i - 1] <= w) {
                dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - weights[i - 1]] +
values[i - 1]);
            } else {
                dp[i][w] = dp[i - 1][w];
            }
        }
    }

    return dp[n][W];
}

int main() {
    int W = 50;
    vector<int> weights = {10, 20, 30};
    vector<int> values = {60, 100, 120};
    int n = weights.size();

    cout << "Maximum value in Knapsack = " << knapsack(W, weights, values,
n) << endl;

    return 0;
}
```



```
└─$ cd "/mnt/data/home/tawheed/Documents/Progra
cuments/Programming/DAA lab/"01knapsack
Maximum value in Knapsack = 220
```

Matrix multiplication by starssen method

```
#include <iostream>
#include <vector>
#include <limits.h>

using namespace std;

int matrixChainOrder(vector<int> &p, int n) {
    vector<vector<int>> m(n, vector<int>(n, 0));

    for (int L = 2; L < n; L++) {
        for (int i = 1; i < n - L + 1; i++) {
            int j = i + L - 1;
            m[i][j] = INT_MAX;
            for (int k = i; k <= j - 1; k++) {
                int q = m[i][k] + m[k + 1][j] + p[i - 1] * p[k] * p[j];
                if (q < m[i][j]) {
                    m[i][j] = q;
                }
            }
        }
    }

    return m[1][n - 1];
}

int main() {
    vector<int> p = {1, 2, 3, 4};
    int n = p.size();

    cout << "Minimum number of multiplications is " << matrixChainOrder(p,
n) << endl;

    return 0;
}
```



```
└─$ cd "/mnt/data/home/tawheed/Documents/Programming/DAA lab/" &&
tawheed/Documents/Programming/DAA lab/"starson-mat-mul
Minimum number of multiplications is 18
```